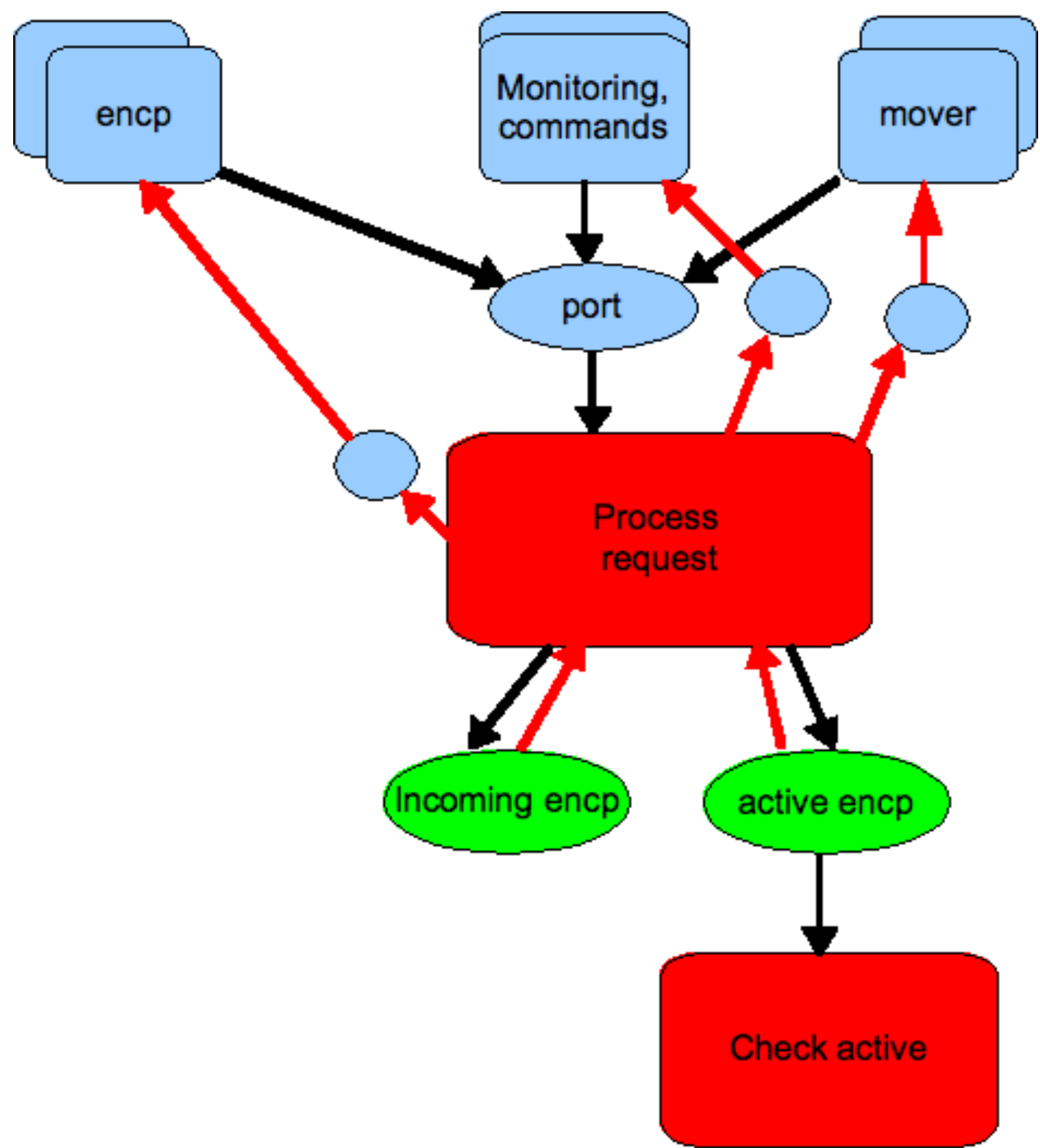# Scalability and Performance Improvements in Fermilab Mass Storage System

Matt Crawford, Catalin Dumitrescu, Dmitry Litvintsev, Alexander Moibenko, Gene Oleynik

Scalable Enstore Request Processing

By 2009 the Fermilab Mass Storage System had encountered several challenges:

- The required amount of data stored and accessed in both tiers of the system (dCache and Enstore) had significantly increased.

- The number of clients accessing Mass Storage System had also increased from tens to hundreds of nodes and from hundreds to thousands of parallel requests.

To address these challenges Enstore and the SRM part of dCache were modified to scale for performance, access rates, and capacity. The diagram at the left represents the request processing of the Library Manager prior to scaling improvements.

The following performance issues were identified:

- Server implementations were single threaded. This was in part the result of the version of python in use, which did not have a threading implementation that could utilize multiple cores.
- In particular, a single thread was used to process different types of requests and was a bottleneck.
- The end user had the ability to specify the delta time for queue priority advancement per request. This feature put a heavy load on CPU resources, unnecessarily sorting request queues even though the feature was not in use.
- The algorithm that throttled transfers from hosts to make sure they did not oversubscribe network resources and hence tie up tape resources was not optimal

→ request
→ reply (normal or error)

The Diagram on the right illustrates the re-architected Library Manager component of Enstore. Improvements in the Library manager included:

- Separate queues and threads for user (encp) requests, monitoring requests and tape drive mover requests using Python multiprocessing threads that takes advantage of multi-core computer architecture
- Removal of the per request queue priority delta time, dramatically speeding up queue priority advancement processing
- The algorithm for throttling the number of active transfers that a given host could have outstanding was improved.

In addition, other Enstore components were made multithreaded where appropriate.

This work increased the number of simultaneously processed requests in a single Enstore Library instance from about 1000 to over 30000. The rates of incoming request to Enstore increased from tens to hundreds per second. Before the improvements, Enstore could not cope with peak CMS loads, resulting in backlogs of requests. Since this work, there have been no requests backlogs under the highest loads CMS has delivered.
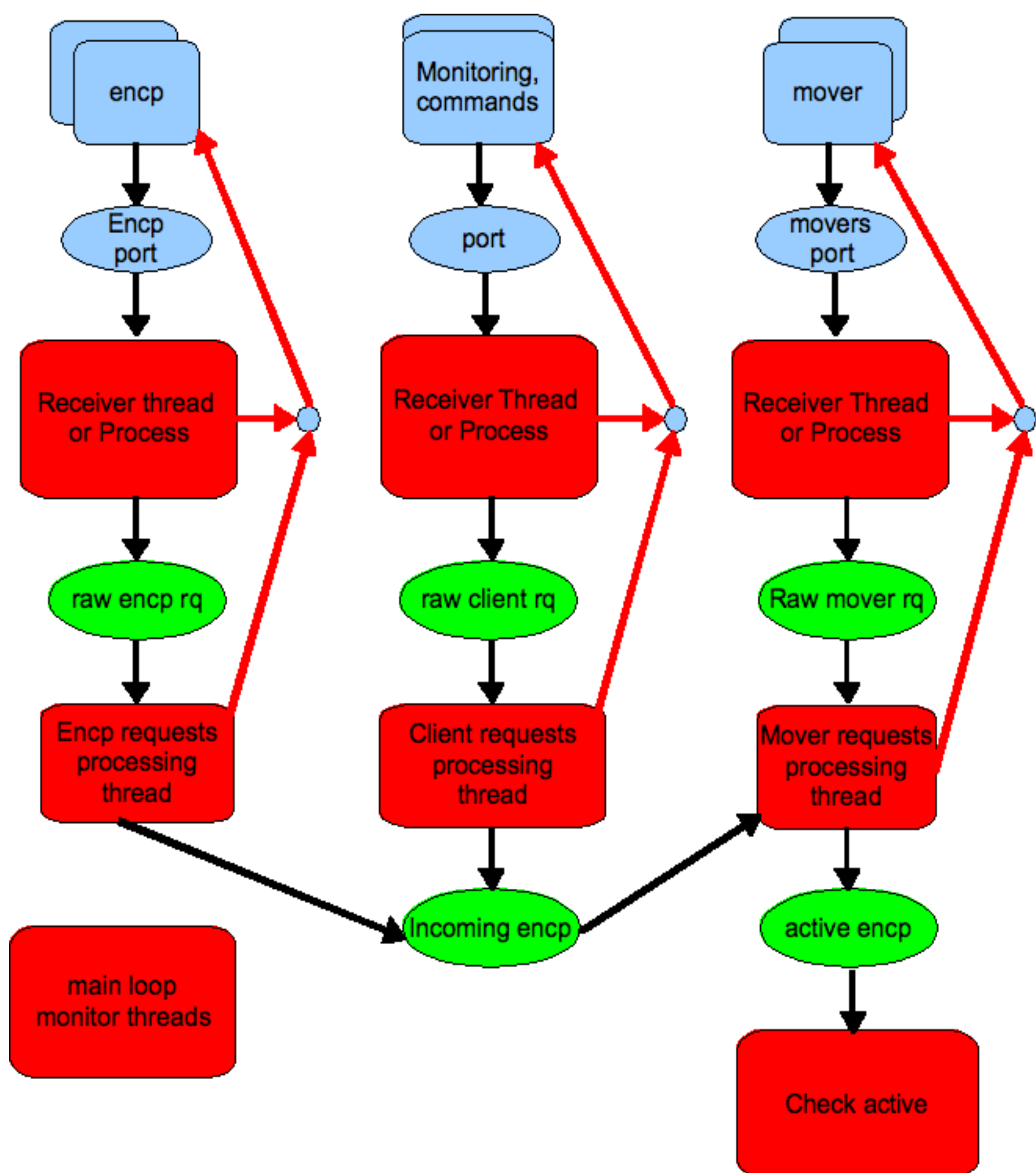

Scalable dCache SRM Server
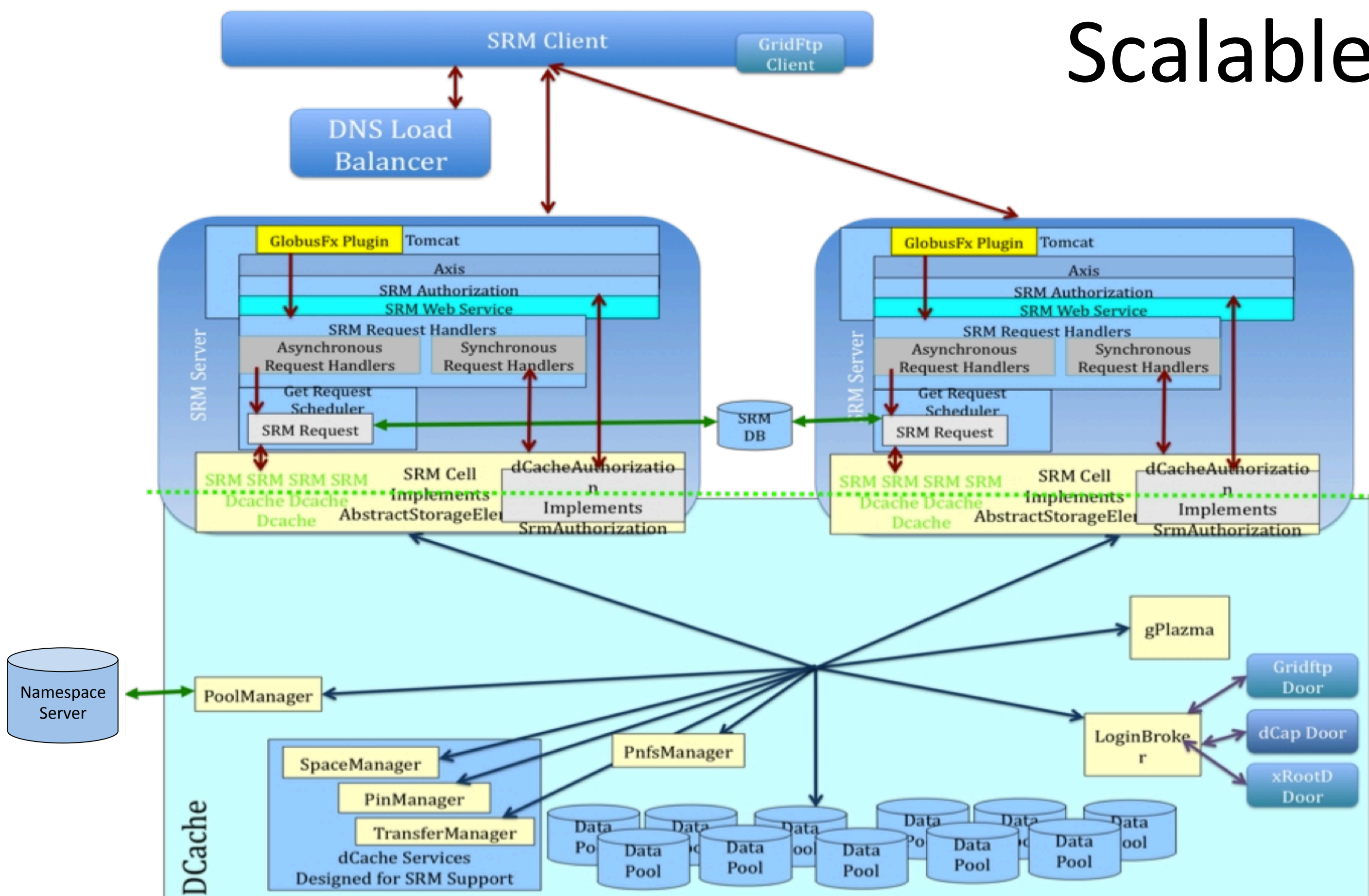
SRM Design
Allow multiple instances of SRM servers run in the system. They share a common database backend for request persistency in DB. The instance of SRM that received the request serves it. Other SRM instances of provide the status of this request by restoring it from the database without caching. Any action on request starts with pulling request status from the DB. Each SRM instance periodically polls in-memory requests to re-check their status in DB to catch request status changes that might have been processed by the instance that actually did not schedule the request.

Testing
The plot shows request processing frequency vs number of simultaneous clients for srmput and srmget operations when running 4 SRM instances. The result is compared to terracotta based scalability solution which was contemplated in 2010 based on SRM code and performance review. Terracotta behaved poorly because of latencies introduced by global memory locks. This is not an ultimate judgment of Terracotta distributed memory management, but rather a testament to the fact that SRM code as designed is not easy to adapt to use with Terracotta.
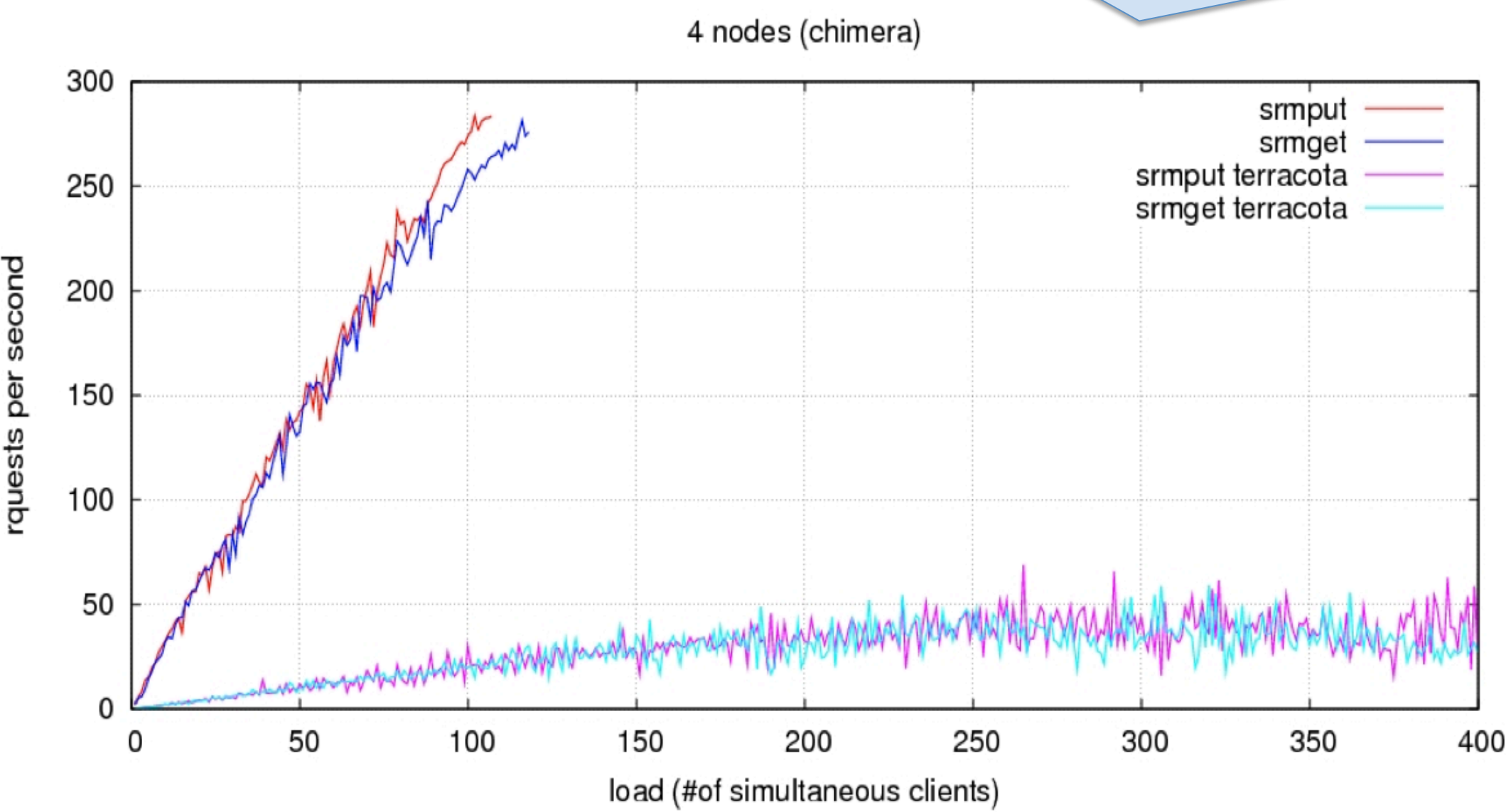
CMS T1 SRM setup
CMS T1 uses round-robin DNS setup.. That is, "cmssrm" resolves to "cmssrm1" or "cmssrm2" on round-robin basis. cmssrm1 and cmssrm2 have their own grid host identities.